Final Report:

Voxel-based Immersive Environments

(31-May-2000)

Sponsored by

Defense Advanced Research Projects Agency (DOD)
(ISO)
ARPA Order D611/70
Issued by U.S. Army Aviation and Missile Command Under
Contract No. DAAH01-00-C-R058

Name of Contractor:	Zaxel Systems, Inc.
Business Address:	10 40th Street, Pittsburgh, PA 15201
Principal Investigator:	Peter W. Rander
Phone Number:	(412) 682-2862

Short Title of Work:	Immersive Environments
Effective Date of Contract:	09-NOV-1999
Contract Expiration Date:	05-JUL-2000
Reporting Period:	09-FEB-2000 till 07-MAY-2000

Summary

This report summarizes the Phase I developments of Zaxel Systems, Inc on award DAAH01-00-C-R058, Voxel-based Immersive Environments. The overall objective of Phase I was to demonstrate the feasibility of real-time voxel-based immersive visualization by achieving four objectives:

- Demonstrate voxel-based 3D reconstruction at 3 frames per second
- Demonstrate interactive view generation at 3 frames per second
- Determine communication bandwidth requirements and potential compression strategies
- Understand complexity of adding audio

Zaxel successfully achieved each of these objectives. As part of the resulting system, we developed a collection of additional hardware and software to address technical requirements that must be met for the algorithms to work successfully. These include

- Multi-camera, multi-PC video capture with recording capability: Required synchronization of video across multiple PCs, as well as extensive efforts to support streaming of video to hard disk. Recording video is important for accurately comparing different algorithms and implementations to one another.
- Camera calibration: Developed calibration object, procedure, and software to determine 3D relationships among cameras. This capability is necessary to fuse information from multiple viewpoints.
- Dedicated Studio: Designed a test-bed studio for immersive communication. This test-bed provides the flexibility to test multiple ideas regarding camera placement and background appearance. It also provides valuable experience in designing and constructing more advanced studios.

1. Identification and Significance of the Problem or Opportunity

For Phase I, we addressed technological needs to develop a real-time 3D reconstruction, transmission, and immersive audio-video presentation system for "virtual" teleportation. Development of such a system would greatly improve the realism of remote collaboration, dissolving the boundaries between the local and remote physical worlds and between the virtual and physical environments.

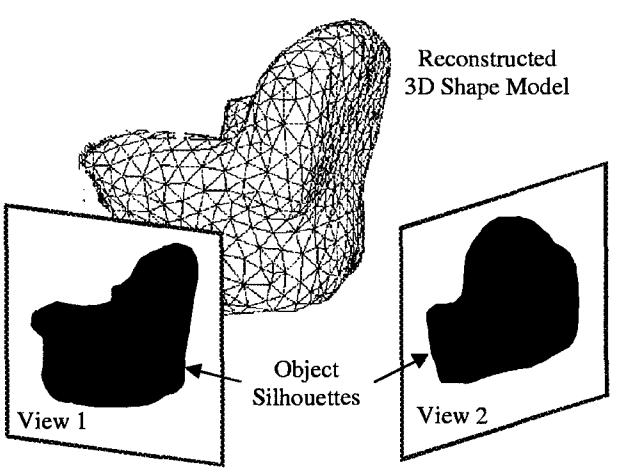
Blurring these boundaries would have immediate value in a variety of applications. In military training, by making shared virtual environments as compelling as real events, trainee decisions and reactions will more accurately reflect true combat responses. As another example, in live combat situations, military leaders will be able to communicate more effectively with personnel on the front lines, especially enhancing the fidelity of nonverbal cues that subordinates may express during exchanges with higher-ranking officers. In addition to training and remote collaboration, non-military applications also include entertainment, such as sports replays or live broadcast, and the creation of special effects for motion.

1.1. Background

Previous research in 3D reconstruction of remote environments showed that it is possible to recover both object appearance and auditory stimuli in remote environments. At the Virtualized Reality lab at CMU, for example, Dr. Rander (co-PI) and Dr. Kanade demonstrated that the combination of image-based stereo and volumetric integration can produce high-quality 3D models [4][5][6][7]. These methods for modeling object appearance, however, consume enormous computational and communication resources -- running on 17 Intel® Pentium II®-based PCs connected by a 10baseT Ethernet hub, the processing required approximately 1000 seconds to process a single second of video. These traits made it appear as if real-time application of these algorithms was nearly impossible in the near future.

Recent developments in the field, however, have begun to greatly reduce the processing time to recover 3D structure. It has been observed that, for human perception of quality, the identification of occluding contours is far more important than precise estimation of smooth surface structure [7]. With this insight, later efforts at CMU have focussed on recovering 3D scene structure directly from the object silhouettes themselves. This process, depicted in FIGURE 1.1, begins by extracting object silhouettes from the input images. These silhouettes are directly integrated in 3D to recover a 3D model of scene structure.

FIGURE 1.1: Silhouette-based 3D model reconstruction.



This process bears some superficial resemblance to the earlier CMU work of using stereo to compute dense range images and then using integration to get a 3D model. In the new algorithm, however, the reconstruction process estimates correspondences only at silhouette boundaries. In addition, the correspondence estimation occurs directly in the volume rather than using the intermediate representation of a range image. The two distinct advantages that this new algorithm offers are much lower computational cost and more realistic view generation.

In light of these advances, Zaxel recognized the possibility of applying this new algorithm to real-time immersive environments.

1.2. Phase I Approach to Remote Immersive Communication

Our Phase I proposal was built on developing and improving the above silhouette-based voxel reconstruction algorithm, while combining it with image-based rendering techniques to create video from virtual viewpoints. FIGURE 1.2 shows a block diagram of the system. Reconstruction from several cameras at one end generates multiple video streams and a 3D model sequence. This information is then used to generate novel viewpoints using video-based rendering techniques.

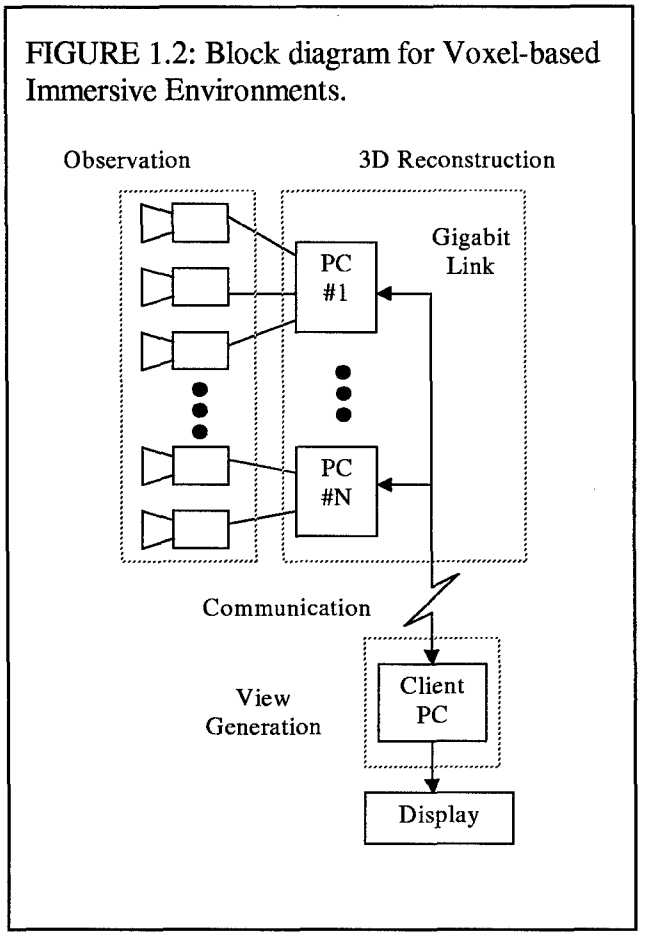


FIGURE 1.2: Block diagram for Voxel-based Immersive Environments.

1.3. Phase I Results

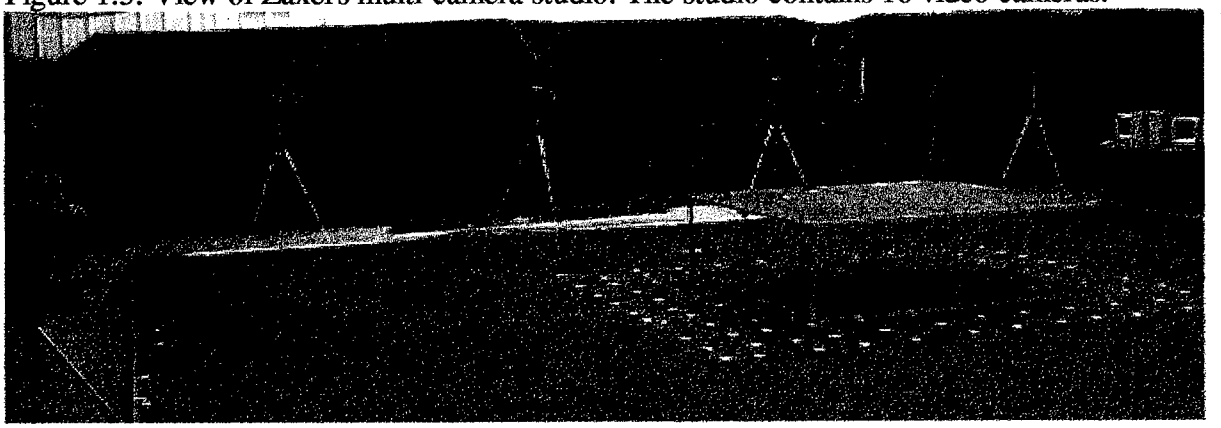
We developed a proof-of-concept system that demonstrated the ability to reconstruct and immersively display real scenes at interactive frame rates with communication over a gigabit Ethernet. In addition, our research uncovered an innovative way to simultaneously improve quality and speed of both reconstruction and rendering for video-only immersive environments.

1.3.1. Attaining Phase I Objectives

The overall objective of Phase I was to demonstrate the feasibility of computational speedup for real-time voxel-based immersive visualization. During the Phase I effort, we achieved all four of the specific objectives laid out in the Phase I proposal:

- Demonstrate basic voxel-based 3D reconstruction: 3 frames per second
- Demonstrate interactive view generation: 3 frames per second

Figure 1.3: View of Zaxel's multi-camera studio. The studio contains 16 video cameras.



- Understand complexity of adding audio
- Determine communication bandwidth requirements and potential compression strategies

As part of the resulting system, we developed additional hardware and software to address technical requirements that must be met for the algorithms to work successfully. These include

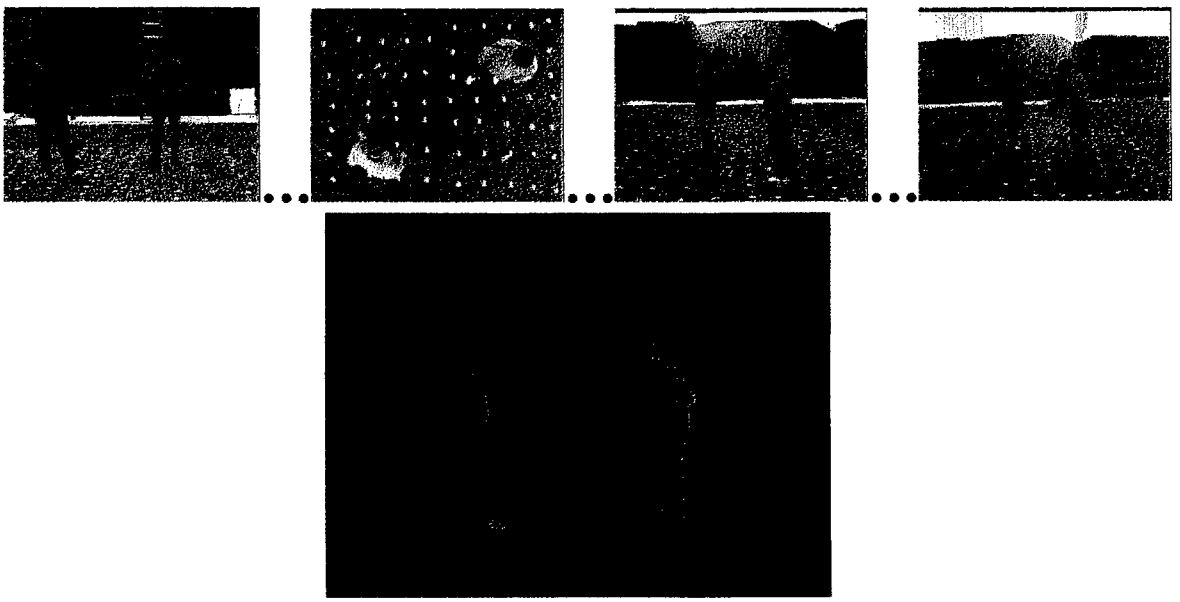
- Multi-camera, multi-PC video capture with recording capability: Required synchronization of video across multiple PCs, as well as extensive efforts to support streaming of video to hard disk. Recording video is important for accurately comparing different algorithms and implementations to one another.
- Camera calibration: Developed calibration object, procedure, and software to determine 3D relationships among cameras. This capability is necessary to fuse information from multiple viewpoints.
- Dedicated Studio: Designed a test-bed studio for immersive communication. Once built, this test-bed will provide the flexibility to test multiple ideas regarding camera placement and background appearance. It will also provide valuable experience in designing and constructing more advanced studios.

FIGURE 1.3 shows a view of the studio we developed for Phase I experimentation. The studio contains 8 PCs each connected to 2 cameras, and 1 additional PC for control and display. The PCs are networked via gigabit Ethernet. The PCs also can stream video to disk in real time at full resolution (640x480 @ 30 fps).

1.3.2. Improving Reconstruction and Rendering

We researched many different rendering techniques in order to find more efficient ways to generate new views of the scene from arbitrary viewpoints. The techniques fall into two broad categories: 1) building a texture-mapped 3D model and 2) Image-Based Rendering (IBR). IBR has been shown to produce image quality that is comparable to the source imagery, especially

FIGURE 1.4: Example of new rendering algorithm. The top row shows 4 of the 15 real images. The bottom row shows an example output image.



when the desired viewpoint is near one of the real cameras. Since our focus is on producing the highest possible image quality, we limited our research to IBR techniques.

Almost all of the existing IBR techniques require that a depth map or a set of correspondences be provided for each real camera view. This depth map provides the distance from the camera to the point being viewed at each pixel. This information is certainly available in the voxel model that we derive from the image silhouettes, but the transformation from voxels to depth maps is not straightforward and is relatively computationally intensive. Since these depth maps would be needed to perform rendering of each frame, they would either need to be precomputed on the server and transmitted to the client (which would entail a large increase in bandwidth), or they would have to be computed at the renderer (with a high computational cost).

This need to compute depth maps from voxel models led us to consider if it is possible to generate depth maps directly from the image silhouettes, skipping the intermediate voxel model. After expending some research effort in this direction, we have developed an algorithm for generating new views of the scene using the implicit 3D model provided by the silhouette masks. An example output of the algorithm is shown in FIGURE 1.4. This method has the following technical advantages:

- silhouettes have about the same size as the voxel model, so similar transmission costs
- depth information can be derived in a computationally efficient manner on the client end
- the resulting model is more accurate than a voxel model

- the method avoids unneeded computation, since only the relevant parts of the 3D model are constructed as they are used

- a depth map and rendered image are computed simultaneously

- a depth map from the perspective of the virtual camera is generated; this can be used for depth cueing (e.g. inserting simulated objects into the environment)

- the method easily handles detection and compensation for object occlusion

One disadvantage of the method is that the transmission cost increases linearly with the number of cameras, versus a fixed cost for the voxel model. With our current system, the two methods have about the same bandwidth requirement with 8 cameras. An additional consideration is that the voxel model has 3D spatial structure, which can be exploited in compression algorithms, whereas the silhouettes have the same amount of data but only 2D structure. More effort may be required to fit the same amount of data into a given bandwidth.

2. 3D Reconstruction

One of the tasks in realizing voxel-based immersive environments is the reconstruction of the voxels themselves. Figure 1.1 shows an overview of this algorithm, which uses multiple images, captured simultaneously, to compute silhouettes -- binary images in which "foreground" structure has one pixel value, and "background" structure has a different pixel value. These silhouettes are directly integrated in 3D to recover a voxel model of scene structure.

In the process of exploring image-based rendering algorithms, we discovered a way to render directly from silhouettes. As we will discuss in Section 3, Rendering, this algorithm not only avoids the extra computation of recovering an explicit 3D model but also produces higher quality renderings. As a result of this discovery, we no longer addressed the needs of explicit voxel reconstruction. For completeness, however, this report includes a description of the voxel reconstruction algorithm.

Whether or not an explicit voxel model is computed, the reconstruction process requires synchronized multi-camera video capture, multi-PC communication, and camera calibration. During Phase I of this project, Zaxel has addressed each of these tasks, providing the core infrastructure required for event reconstruction. Zaxel has also developed a novel silhouette extraction algorithm, which models noise, color, and shadows to create high quality silhouettes.

2.1. Multi-Camera Video Capture and Multi-PC communication

The voxel reconstruction algorithm needs video images captured simultaneously from multiple viewpoints. While video capture of a single video stream has become commonplace in off-the-shelf systems, the capture of multiple video streams is much more challenging, requiring the use of multiple computers, which then requires synchronization among the PCs. We have designed an expandable system that captures video from 2 cameras per PC with an arbitrary number of PCs. We have implemented our design on an 8-PC cluster with 16 cameras, controlled by one additional PC. Each PC contains a RAID subsystem for storage of multi-camera video. The ability to store the video is valuable for reliably comparing and tuning the performance of processing algorithms.

To address synchronization, we use two hardware subsystems and custom video capture software. First, a video synchronization (sync) generator and a set of video distribution amplifiers are used to drive each camera. The cameras adjust the opening of their shutters to match the timing of the incoming sync signal. Next, we use a video time code generator to produce a video time code. The resulting time code is copied and inserted into each video stream as a Vertical Interval Time Code (VITC). Finally, we adjust the digitizer to capture the VITC as additional lines in each image, and then apply a custom software detection algorithm to interpret the VITC. As a result, each PC can determine which video frame is being processed, and can tag each resulting image appropriately. Since all PCs have the same reference, they all produce the same tag for images captured at the same time.

The last major piece of the video capture system is the communications mechanism to control and communicate data among PCs. We designed and implemented a mechanism, called zComm,

which provides this functionality for an arbitrary number of PCs. This design creates a "virtual machine" that provides ways to access data independent of its location -- in the same process, in a different process on the same machine, or on a different machine. zComm also remotely starts, stops, and monitors software on the PCs in the cluster, providing complete infrastructure for distributed processing.

2.2. Camera Calibration

3D reconstruction and rendering require a mapping between each image and a common 3D coordinate system. The process of estimating this mapping is called camera calibration. Each camera in a multi-camera system must be calibrated, requiring a multi-camera calibration process. The mapping between one camera and the 3D world can be approximated by an 11-parameter camera model, with parameters for camera position (3) and orientation (3), focal length (1), aspect ratio (1), image center (2), and lens distortion (1). Camera calibration estimates these 11 parameters for each camera.

The estimation process itself applies a non-linear minimization technique to samples of the image-3D mapping. To acquire these samples, an object must be precisely placed in a set of known 3D positions, and then the position of the object in each image must be computed. This process requires a calibration object, a way to precisely position the object in the scene, and a method to find the object in each image. For a calibration object, we designed and built a calibration plane approximately 2.5 meters by 2.5 meters, which can be precisely elevated to 5 different heights. The plane itself has 64 LEDs laid out in an 8x8 grid, 30 cm between each LED. The LEDs are activated one at a time so that any video image of the plane will have a single bright spot in the image. By capturing 64 images from each camera, each LED is imaged once by each camera. By sequencing the LEDs in a known order, software can determine the precise 3D position of the LED. Finally, by elevating the plane to different heights, a set of points in 3 dimensions can be acquired. Once all the images are captured, a custom software system extracts the positions of the LEDs in all the images and then applies the calibration algorithm. The operator can see the accuracy of the camera model, and can compare across cameras. The operator can also remove any LEDs that are not properly detected by the automated system.

2.3. Silhouette Construction

As shown in Figure 1.1, the reconstruction algorithm uses multiple images, captured simultaneously, to compute silhouettes -- binary images in which "foreground" structure has one pixel value, and "background" structure has a different pixel value. Zaxel Systems, Inc. has developed a novel silhouette extraction algorithm that models the colors of background pixels by a Gaussian distribution. Scene pixels are compared with the pixels in the background. If the scene pixel fits the background Gaussian distribution for the corresponding background pixel, then it will be marked as background. Otherwise it will be marked as foreground.

2.3.1. Gaussian statistic model

The Gaussian statistical model can be written as the following formula:

$$g(X)=\frac{1}{(\sqrt{2\pi})^n(\sqrt{|\det(V)|}}e^{\frac{(X-M)^T V^{-1}(X-M)}{2}}, \quad (1)$$

where X is a variation in $\Re^n$, M is the mean of X, and V is the covariance matrix.

2.3.2. Background model

For the foreground/background segmentation application, we assume the distribution of intensity of each pixel in the background is a Gaussian distribution. And we choice to represent image in (*yuv*) color space, because it is easier to eliminate shadow. The relationship between (*yuv*) and (*rgb*) color space can be described by the following linear transform:

$$y = 0.257r + 0.504g + 0.098b + 0.063$$
$$u = -0.148r - 0.291g + 0.439b + 0.500$$
$$v = 0.439r - 0.368g - 0.072b + 0.500$$

Assume the independence among y, u and v, the Gaussian function (1) becomes:

$$g(X)=\frac{1}{(2\pi)^{\frac{3}{2}}\sigma_y\sigma_u\sigma_v}e^{-\frac{1}{2}\left\{\frac{(y-m_y)^2}{\sigma^2{}_y}+\frac{(u-m_u)^2}{\sigma_u{}^2}+\frac{(v-m_v)^2}{\sigma_v{}^2}\right\}}. \quad (2)$$

The probability density $g(x)$ is used to classify the pixels as foreground or background. We select a global threshold *TH* for all pixels by experiments to achieve a desired percentage of correct detection of background. For each pixel x, if $g(x) > TH$, this pixel is considered as the background, otherwise it is the foreground.

This algorithm is very sensitive to any small change of pixel's intensity. But at the same time, some pixels in shadowed background are also marked as foreground. To eliminate these shadows, we need to change the function (2) so that the shadowed background pixels are not classified as foreground.

2.3.3. Shadow elimination

The main reason that the pixels in shadowed background were detected as foreground is illumination change of these pixels. The underlying color of the background, however, is unchanged. If color information is more directly exploited, the shadow will be suppressed. So we

scale chromaticity components u and v by y, and represent them by $p=u/y$ and $q=v/y$. So the equation (2) is redefined as:

$$g(X) = \frac{1}{2\pi\ \sigma_p \sigma_q} e^{-\frac{1}{2}\{\frac{(p-m_p)^2}{\sigma_p^2}+\frac{(q-m_q)^2}{\sigma_q^2}\}} \quad . \quad (3)$$

The p and q are more robust to illumination changes from shadows or highlights. Equation (3) works very well when the foreground is colorful.

However, if the foreground has little color -- for example, a white object in a gray background -- then the object is considered as background. In order to get better results, y still needs to be used. The function (3) comes to (4):

$$g(X) = \frac{1}{(2\pi)^{\frac{3}{2}} \sigma_y \sigma_p \sigma_q} e^{-\frac{1}{2}\{\frac{(y-m_y)^2}{\sigma^2_y}+\frac{(p-m_p)^2}{\sigma_p^2}+\frac{(q-m_q)^2}{\sigma_q^2}\}} \quad . (4)$$

Now for each pixel in scene image, we subtract y by an offset, say *lowoffset*. Search *lowoffset* in range [0,*low*], where *low* is the maximum illumination change allowed in shadowed background. If there is at least one y, so that $g(x)>TH$, this pixel is marked as background. Otherwise, we must do more check. Add y with an offset, say *highoffset,* and search *highoffset* in range [*0,high*], where *high* is the maximum illumination change allowed in highlight background. If there is at least one y, so that $g(x)>TH$, this pixel is marked as background, else it is foreground.

2.3.4. Speed up

If formula (4) is used directly to classify, it is time consuming. So we can eliminate some complicated computing in formula (4). First, apply log to both side of formula (4),

$$\ln(g(x)) = -1.5\ln(2\pi) - \ln(\sigma_y \sigma_p \sigma_q) - \frac{1}{2}\frac{(y-m_y)^2}{\sigma_y^2} - \frac{1}{2}\frac{(p-m_p)^2}{\sigma_p^2} - \frac{1}{2}\frac{(q-m_q)^2}{\sigma_q^2} \quad .$$

For a pixel (y,u,v), convert it to (y,p,q). If it belongs to background, then the following is true:

$$-1.5\ln(2\pi) - \ln(\sigma_y \sigma_p \sigma_q) - \frac{1}{2}\frac{(y-m_y)^2}{\sigma_y^2} - \frac{1}{2}\frac{(p-m_p)^2}{\sigma_p^2} - \frac{1}{2}\frac{(q-m_q)^2}{\sigma_q^2} > \ln(TH) \quad . \quad (5)$$

The formula (5) can be written as:

$$(y-m_y)^2 < \{-2\ln(TH) - 3\ln(2\pi) - 2\ln(\sigma_y \sigma_p \sigma_q) - \frac{(p-m_p)^2}{\sigma_p^2} - \frac{(q-m_q)^2}{\sigma_q^2}\}\sigma_y^2 \quad (6)$$

Let $A = \{-2\ln(TH) - 3\ln(2\pi) - 2\ln(\sigma_y\sigma_p\sigma_q) - \frac{(p-m_p)^2}{\sigma_p^2} - \frac{(q-m_q)^2}{\sigma_q^2}\}\sigma_y^2$,

So formula (5) becomes:

$$(y-m_y)^2 < A. \qquad (7)$$

In equation (7), if A<0, the formula is always false, so this pixel will be marked as foreground. If A>0, we need to do the test which was described in section 3. Instead of testing each y, it can be replaced by checking whether y lies in range $[m_y - \sqrt{A} - low, \sqrt{A} + m_y + high]$. So, the decision procedure is:

(1) If $A < 0$, it's foreground; else

(2) If $y \le \sqrt{A} + m_y + high$, and $y \ge m_y - \sqrt{A} - low$, it's background, otherwise it's foreground.

For a special background, part of A is constant. So before starting to process a sequence of scene, pre-compute the expression $-2ln(TH) - 3\ln(2\pi) - 2\ln(\sigma_y\sigma_p\sigma_q)$ for each pixel of background. This makes the algorithm much faster. We also employed parallel computing technique by implementing the algorithm in Intel PIII architecture.

2.3.5. Sigma correction

One important issue that must be taken into account is that some pixel' σ_y, σ_p or σ_q are very small, even equal to 0. This is partly because of saturation in the input video. In this situation, foreground/background segmentation may fail. To prevent this condition, σ_y, σ_p and σ_q should be set to at least a minimum threshold. Considering that the digitizer also has ½ bit maximum error, we restrict σ_y, σ_p and σ_q to be at least 0.5.

Increasing σ_y, σ_p and σ_q will make the algorithm less sensitive to pixels' intensity change. Specially, if there are some saturated areas in background, or there is significant reflection from object to background, adjusting σ_y, σ_p and σ_q can get better result.

2.3.6. Mathematics morphological filter

After the above processing, a background mask is generated. But there are still some false detection points in mask, such as holes in the foreground or isolated spots that actually belong to background. A binary morphological filter is applied to the mask to deal with this problem. The morphological filter consists of a sequence of six mathematical morphological operations. First a 3x3 dilate operation followed by 3x3 erode operation fill the small holes. Then a 5x5 erode followed by a 5x5 dilate delete the isolated spots. In the last stage, a MxM dilate operation is followed by a MxM erode operation to fill large holes. The window size of the filter is selected

based on image size, the object size, and how the boundary quality. In our experiment, if input image is 640x480, we select the window size M=11. If the input image is 320x240, we select window size M of 5 or 7. In order to aim for real-time application, the morphological filter has implementation of Intel PIII.

2.4. Silhouette-based Voxel Reconstruction

As originally proposed, an explicit 3D voxel model is computed from a set of silhouettes and the camera parameters (position, orientation, focal length, etc) for each viewpoint. This section describes the voxel computation algorithm. Note, however, that in the course of our research, we discovered an innovative way to render without computing an explicit voxel representation. The rendering algorithm is presented in section 3, Rendering.

2.4.1. 3D Voxel Reconstruction

The voxel reconstruction algorithm works by carving away voxels that are part of the background. If the projection of a voxel lands in a region of the silhouette that has no foreground labels, then the algorithm immediately classifies the voxel as empty, skipping the evaluation of any remaining images. The algorithm labels a voxel as occupied only when the "background" test fails for each image. (This condition is equivalent to say that each projection contains at least one silhouette pixel labeled foreground.) The resulting voxel model is guaranteed to contain no less than the full 3D structure represented by the silhouettes -- what we term a "conservative" reconstruction algorithm.

This algorithm can easily be modified to support multi-resolution voxel processing. In particular, the algorithm can easily be written using the oct-tree volume representation. An oct-tree is a 3D spatial data structure in which space (or volume) is decomposed into 8 children -- permutations of three 1-D decompositions: top vs. bottom, left vs. right, front vs. back. Each child can be divided in the same way, creating 8 children each with 8 children. The process can be coded easily using recursive data types and self-referential functions. The recursion stops when a sufficient level of detail is reached. The pseudo-code changes slightly with this multi-resolution framework, because the recursion should be performed only in space near the surfaces in the scene.

3. Rendering

The CMU Virtualized Reality project previously used a 3D model construction from stereo followed by triangularization. The models thus produced were texture-mapped with the original images. Such texture-mapped polygons can be easily rendered with even low-end 3D accelerated hardware. While this method lends itself well to modern hardware, the quality of the resulting images was low due to the coarse sampling of both the 3D information and the textures.

We researched many different rendering techniques in order to more efficiently generate new views of the scene from arbitrary viewpoints. The techniques fall into two broad categories: 1) building a texture-mapped 3D model and 2) Image-Based Rendering (IBR). IBR has been shown to produce image quality that is comparable to the source imagery, especially when the desired viewpoint is near one of the real cameras. Since our focus is on producing the highest possible image quality, we limited our research to IBR techniques.

In order to improve the quality of reconstructed images, we decided to use a method conceptually based on Image-Based Rendering (IBR). There are two basic types of IBR methods:

1. Forward projection of reference image pixels onto the 3D voxel model and then into the virtual image.

2. Inverse projection of pixels from the virtual image onto a 3D voxel model and then back into the reference images.

Because there are a number of benefits and few problems with the inverse projection method, we decided to implement it for our Phase I system.

3.1. Inverse Projection

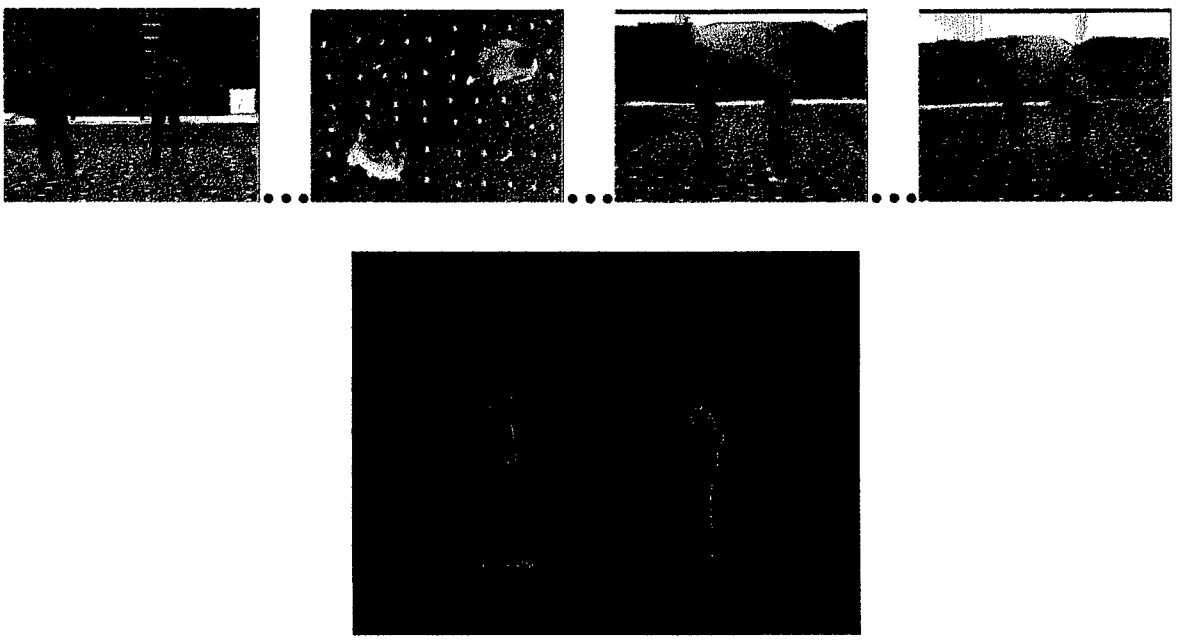
This method involves looping over the pixels in the virtual image. Each pixel is projected onto the 3D voxel model and then projected into each of the reference views in turn. The path between the model and each of the reference cameras must be explicitly checked for occluding objects.

Inverse projection is a less frequently used technique than forward projection methods, though it has obvious advantages in that each pixel of the final image is automatically given the proper color. It is discussed in [1] and briefly in [3].

3.2. Elimination of the Explicit Voxel Model

Almost all of the existing IBR techniques require that a depth map or a set of correspondences be provided for each real camera view. This depth map provides the distance from the camera to the point being viewed at each pixel. This information is certainly available in the voxel model that we derive from the image silhouettes, but the transformation from voxels to depth maps is not straightforward and is relatively computationally intensive. Since these depth maps would be needed to perform rendering of each frame, they would either need to be precomputed on the

FIGURE 3.1: Example of new rendering algorithm. The top row shows 4 of the 15 real images. The bottom row shows an example output image.



server and transmitted to the client (which would entail a large increase in bandwidth), or they would have to be computed at the renderer (with a high computational cost).

This need to compute depth maps from voxel models led us to consider if it is possible to generate depth maps directly from the image silhouettes, skipping the intermediate voxel model. After expending some research effort in this direction, we have developed an algorithm for generating new views of the scene using the implicit 3D model provided by the silhouette masks. A related algorithm has recently been published by a group at MIT [2].

An example output of the algorithm is shown in FIGURE 3.1.

This method has the following technical advantages:

- silhouettes have about the same size as the voxel model, so similar transmission costs
- depth information can be derived in a computationally efficient manner on the client end
- the resulting model is more accurate than a voxel model
- the method avoids unneeded computation, since only the relevant parts of the 3D model are constructed as they are used
- a depth map and rendered image are computed simultaneously

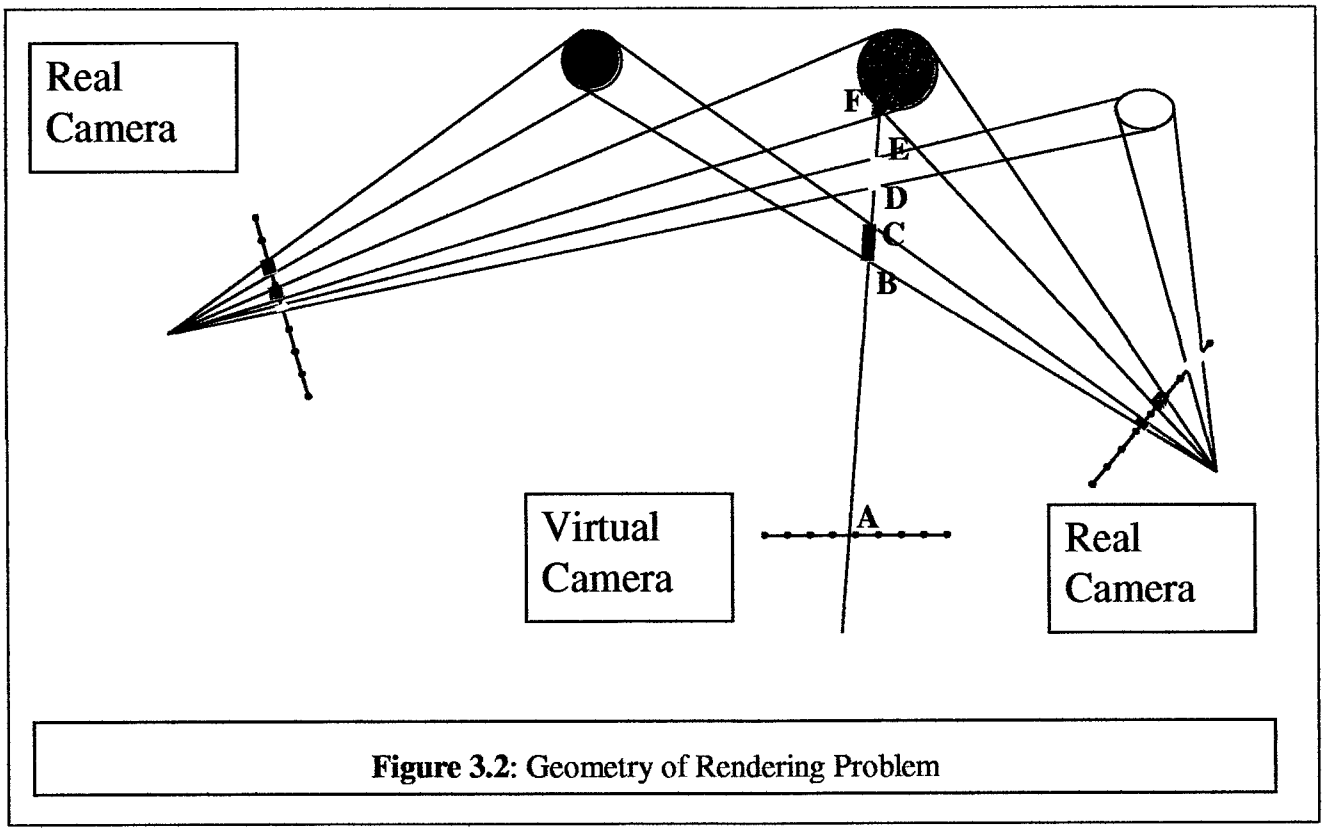


Figure 3.2: Geometry of Rendering Problem

- a depth map from the perspective of the virtual camera is generated; this can be used for depth cueing (e.g. inserting simulated objects into the environment)
- the method easily handles detection and compensation for object occlusion

One disadvantage of the method is that the transmission cost increases linearly with the number of cameras, versus a fixed cost for the voxel model. With our current system, the two methods have about the same bandwidth requirement with 8 cameras. An additional consideration is that the voxel model has 3D spatial structure, which can be exploited in compression algorithms, whereas the silhouettes have the same amount of data but only 2D structure. More effort may be required to fit the same amount of data into a given bandwidth.

3.3. Direct Depth Computation and Rendering Algorithm Concept

3D reconstruction using the voxel intersection method slices away discrete pieces of 3D space that are considered to be unoccupied. When a particular camera sees a background pixel, it is safe to assume that the space between that camera and the background at that pixel is empty. This space is actually shaped like a rectangular pyramid with its tip at the focus of the camera, extending out until it intersects the background.

The key idea here is that if a particular 3D location in space is seen as unoccupied by any one camera, the point will be considered unoccupied regardless of what the other cameras see there.

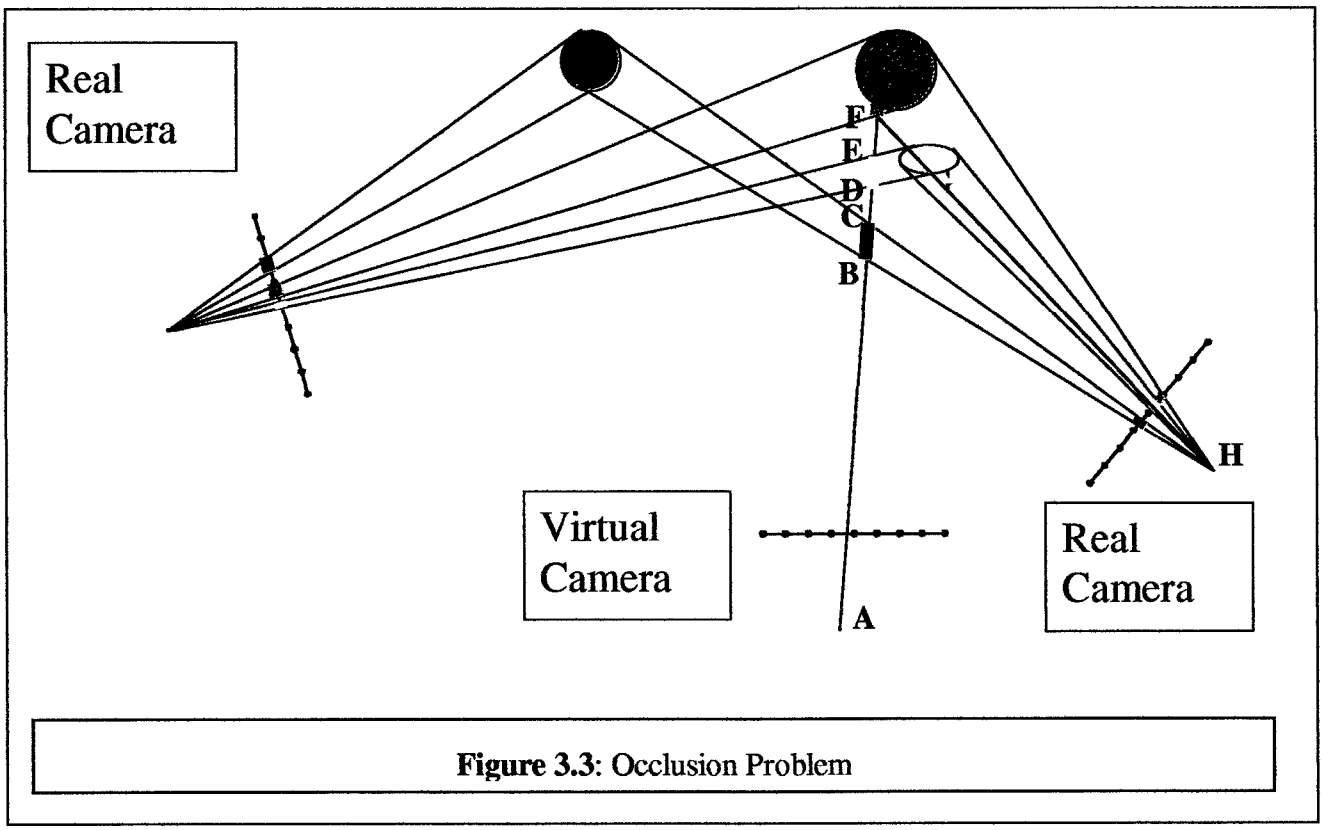


Figure 3.3: Occlusion Problem

For each pixel in the virtual image, we will move a test point out along the ray corresponding to that pixel, as illustrated in Figure 3.2. At each point along the ray, we will evaluate whether the corresponding pixel in each image sees the background. In the example of Figure 3.2, we follow the example ray outward from the point marked A. If any of the cameras sees background at a particular point, we consider that point to be unoccupied, move one step farther out along the ray, and repeat the process. In the example, for each of the points from A to B, no camera considers the points to be occupied. From B to C, the camera on the right sees the red object, but the camera on the left sees nothing. From C to D, again no camera sees anything. From D to E, the camera on the left sees the yellow object, but the camera on the right sees nothing. From E to F again neither camera sees anything. Finally, at F, both cameras agree that the point is occupied and the search stops.

When we have found a 3D point that all cameras agree is occupied, we have found the depth of that pixel, as well as knowing the position of the point in all of the images. In order to render the pixel, we just have to combine the pixels at these points from the real images.

3.4. The Occlusion Problem

Once we have a set of pixels to render in the virtual camera, we must use them to select a color for each virtual camera pixel. One of the biggest problems we can run into is that most of the cameras are not looking at the point that we want to render. For many of the cameras, this is obvious: they are facing in the wrong direction and seeing the back side of the object. But this problem can occur even when cameras are pointing in almost the same direction as the virtual

camera, because of occlusion. In this context, occlusion refers to the situation where another object blocks the view of the object that we must render. An example is shown in Figure 3.3. In this case, the camera on the right cannot see the green object at F, so we should not use its pixel value to render this pixel.

In order to detect occlusions, we apply the following technique. For each camera, we pre-compute a depth map using the same algorithm as for the virtual camera, as described in the previous section. To determine if a pixel from a given camera is occluded in the virtual view, we use the computed depth in the virtual camera (in this case, the distance from A to F) to transform the virtual pixel into the real camera view (computing the distance from H to F). If the depth of the pixel from the virtual view (HF) matches the depth computed for the real view (HG), then the pixel is not occluded and we can use it for rendering. In the case of Figure 3.3, the depths do not match, so we must choose pixels from other cameras.

3.5. Display and GUI

The renderer of course needs an interactive display. The code to do this was written in OpenGL, and provides several options to allow the rendered result image to be scaled to variable window sizes on the display.

We found that it is difficult for the user to judge his virtual position relative to the objects in the world when the objects themselves are moving. For this purpose, we added a static background in order to give feedback to the user about their position relative to the static world.

The current GUI provides several methods for controlling the position of the virtual camera, none of which seem to be intuitive to first-time users. More research in this area is definitely needed.

3.6. Algorithm Speed and Optimizations

The first implementation of this algorithm ran at about 10 seconds per rendered frame. We performed a number of optimizations in order to meet the 3-5 fps required by our Phase I SBIR contract. The optimization process was carried out in a standard fashion. We used a code analysis tool to identify "hot spots" in the code (places where an inordinate amount of time was being spent), and worked to reduce the amount of time spent at each "hot spot". The optimizations performed fell into three broad categories listed below. An example of each is given.

1. Algorithm Optimizations – these are changes to the rendering algorithm that make it run faster. These fall into two broad categories:

    a. Optimizations designed to do less work overall. One of the most important optimizations was to take the bounding box of each real image (this is the smallest axis-aligned rectangle in the image that contains all foreground pixels). All foreground objects must lie in the 3D volume that is the intersection of all of the bounding boxes. This volume can be computed cheaply, and used to decide both

which pixels of the virtual image need to be rendered and what the upper and lower bounds on the search at each pixel should be.

b. Optimizations designed to use less expensive operations. Most operations in the algorithm involving depth really want to use inverse depth (i.e. 1/z instead of z). Since division operations are very expensive on most processors, it helps to reduce the number of divisions that are performed. By moving the division process early into the algorithm and working with inverse depth, the number of divisions per pixel was cut in half

2. Standard Code Optimizations – these optimizations are well known and often used to improve speed, but are often not done by an optimizing compiler. Examples are eliminating branch instructions by combining or removing if-then constructs, eliminating redundant calculations, and replacing multiply-add loops with simple addition operations.

3. Architecture Specific Optimizations – these are optimizations that are specific to the Intel architecture. Examples are replacing the very costly integer truncation on Intel with inline assembly language to round floating point numbers to the nearest integer, and reordering of operations to better suit the pipeline of the processor.

Note that the only code in the renderer that is currently specific to the Intel architecture is the integer round function, thus it would be easy to port the renderer to other architectures.

The renderer does not currently take advantage of the MMX or Streaming SIMD instructions provided by the Pentium III processor. There are many places where these instructions could be used, so this is an area for future improvement.

The following table summarizes the average performance of the current renderer on an 8-camera data set using a 360° rotation.

Processor	320x240 resolution	640x480 resolution
550 MHz Pentium 3	11.16 fps	2.99 fps
800 MHz Pentium 3	18.02 fps	4.93 fps
Dual 500 MHz Pentium 3	22.18 fps	6.19 fps

Recent analysis of the rendering code shows that it is spending about 40% of its time in projection operations from the 3D world space into the 2D image space, and 40% of its time in search along the rays for each pixel. This is somewhat of a surprise, since we expected most of the time to be spent in search. At this point it may be desirable to re-investigate the use of an explicit voxel model. A renderer based on explicit voxel models does not need to do as many 3D to 2D projections.

4. Communication

During the Phase 1 of this project we investigated communication architecture and protocols for immersive environments, devised compression algorithms and implemented them to obtain the statistics of these compression algorithms, and built and integrated capture to render system. We will describe these three achievements.

4.1. Communication Architecture and Protocols

We have investigated communication protocols and compression algorithms to allow viewers from remote locations to choose their own viewpoints freely to experience immersive environments. We have two modes of operations in mind. The first is one-to-one communication where there will be only one viewer. The second is one-to-many communication where an unlimited number of people can choose their own viewpoints

4.1.1. One-to-one communication

The processing of immersive environments falls into the following block diagram (Figure 4.1):

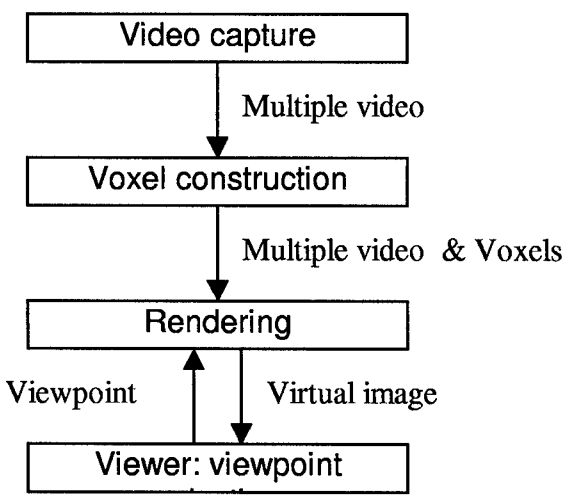


FIGURE 4.1

In one-to-one communication all the functions will be computed at the server, as shown in Figure 4.2. The video is created at the server and sent to the client. The client feeds back the new viewpoint. Since the only information sent from the server to the client is one video stream, the communicationbandwidth requirement is the same as any other video communication on the Internet or on dedicated lines. Therefore, we can use the state of the art video compression technique that best suits the communication pipes. We will use MPEG video compression when sending video over T1 line, whereas for 56K bit Internet connection, we can use one of the more popular streaming video such as Windows Media or Real Video.

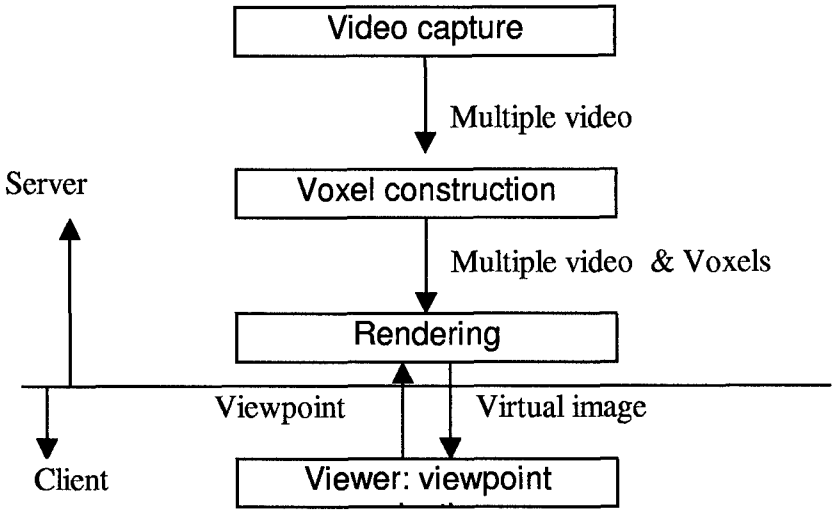


FIGURE 4.2

4.1.2. One-to-many communication

If we apply the algorithm used for the one-to-one communication approach to one-to-many communication, both the computation requirement of the server and the bandwidth requirement of the communication pipes grow linearly as the number of clients since every client has the request to take a different viewpoint. Therefore, we need to take a different approach to one-to-many communication.

The approach we take is to partition the whole processing described in FIGURE 4.1 into two parts. The first part consists of the video capture and voxel construction. The second part consists of the rendering for creation of a virtual viewpoint based on the user feedback. The first part is computed at the server, but the second part is computed at the client as shown in Figure 4.3. Then the data that will be sent across the communication pipes are captured video and voxels. The advantage of this approach is that the computation requirements of the server are constant, since no matter how many viewers are participating, the server needs to perform only video capture and voxel construction. The bandwidth requirement is also constant since every viewer needs to receive only the captured videos and voxels. The user view selection information does not need to be sent back to the server either. The virtual viewpoint creation is performed at each client PC with the feedback from the user, so the response should be as fast as can be. PCs have huge computation power, often as much as the servers have, so the offloading of the rendering operation to the client should not cause any degradation in the quality of the rendered images. Thus, the system scales very well even as the number of viewers grows.

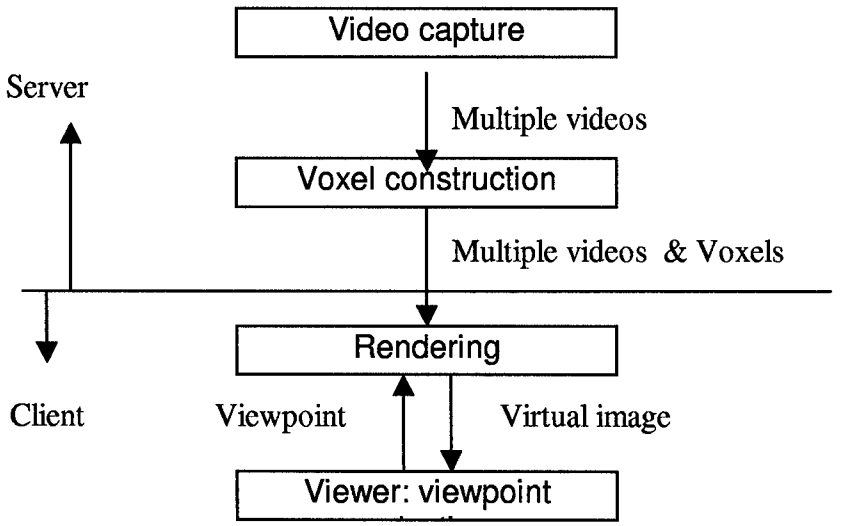


FIGURE 4.3

With this approach thecommunication bandwidth requirement is bounded and will not grow even though the viewers grow. However, the drawback is that the total volume of videos and voxels is still very large. If the application is to broadcast immersive environments within a small area using Ethernet, this architecture and protocol is sufficient. However, if we need to broadcast over WAN, we need to reduce the bandwidth requirement further.

In the rendering algorithm, our experiment showed that three videos that span the new viewpoint is sufficient to create a good image and also to eliminate occlusion. Therefore, we need to send 3 videos and voxels to each client. The server transmits 16 different videos streams and 1 voxel stream. The client selects 3 video streams out of these 16 and 1 voxel stream as the view point is selected. (Figure 4.4)

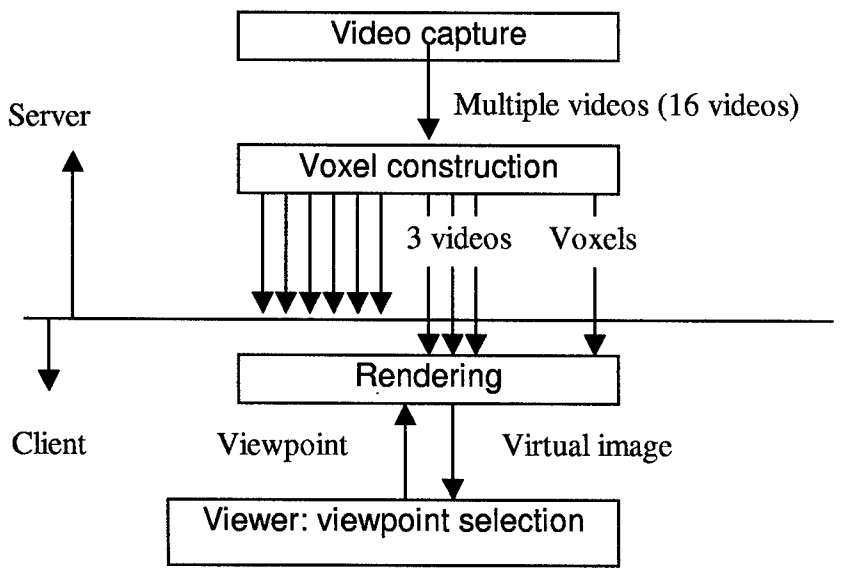
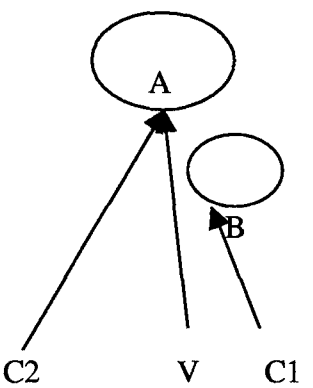


FIGURE 4.4

We would like to further reduce the number of video sent to a client to one video. The most common approach is to send the video stream that is the closest to the virtual view point. This approach works well for many simple object, and the selection algorithm is easy. However, it does not always work well when there are multiple objects because of occlusion. It is shown by an example below. (Figure 4.5)



FUGURE 4.5

When a virtual view point V is observing the point A of the far object, the algorithm selects the closest camera C1 to render A. However, C1 is obstructed the view by the near object and the color for A is picked by the color at point B. In this case if the algorithm selects camera C2 to render point A, it had the right color. This problem is less severe if we have more cameras. If there is a camera C3 in between C1 and C2, then the closest camera to V is C3 and the color selected would have been correct.

Therefore, we partially solve this problem by the method called "auxiliary camera". We synthesize in between views with multiple cameras. We call these views the views of auxiliary cameras. Since we use multiple cameras, the views of auxiliary cameras have less problems of occlusion. Then the final rendering algorithm with one camera is to choose the real camera or auxiliary camera closest to the virtual view point and render the object. (Figure 4.6)

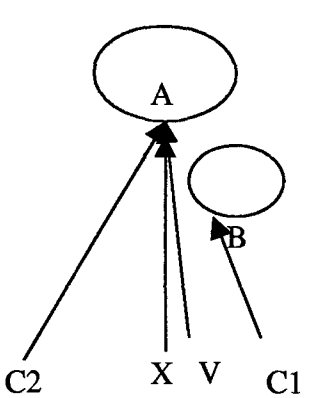


FIGURE 4.6

In this example an auxiliary camera X is created in between C1 and C2. Then the closest camera to V is X, which is looking at the point A and has the right color.

The implication to the server architecture is that the video streams sent out are both videos from real cameras and synthesized videos from auxiliary cameras. The client chooses the closest camera video and renders the voxels with that video.

Then for Internet applications where the bandwidth is limited, each client receives one video stream from the closest virtual camera and voxel data structures. One virtual camera video can be compressed using MPEG or other streaming video compression algorithm. Voxel data structures can be efficiently compressed. The algorithms and compression results are discussed in the next section.

4.2. Compression

As the protocols to broadcast videos and voxels have been established in the previous section, the next issue is to compress data sent over the communication line. The ultimate goal is to send the immersive environment over the 56K bps Internet. Since we need to send audio and video, the bandwidth that we can allocate to voxels or silhouettes is 10% of the total communication bandwidth. If this is achieved, we can use the state of the art video compression algorithms that are typically used for the particular communication line. Voxel or silhouette compression, unlike the compression of video, must be lossless compression.

With our target of 5600 bps for voxel and silhouette data and at 6 models per second for 56K bps Internet, voxels and silhouettes for each frame must be compressed to 900 bps.

4.2.1. Voxel Compression

We performed compression of voxel data structures first by representing them as an octtree, and then representing them as a 3D array of bits.

Octtree Compression

Algorithm 1: Simple coding

Octtree is represented as a tree of 8 siblings. Each node is occupied, vacant, or internal node. Because of the frequency of these nodes in the octtrees we examined, we present occupied as 0, vacant as 10, and internal node as 11.

Algorithm 2: Runlength coding

Runlength coding is applied to the result of Algorithm 1.

Algorithm 3: Runlength with Huffman coding

Huffman coding is applied to the result of Algorithm 2.

Algorithm 4: Difference coding

Taking the difference between two octtrees corresponding to successive frames and apply codes to the difference octtree in the following rule: 0 – no change, 10 – internal node, 110 – change from vacant to occupied, 111 – change from occupied to vacant.

Measurement

We compressed voxel data structures of 64X64X64, whose file size is 32 M Bytes, and 32X32X32, whose file size is 4 M Bytes. The result is shown in the following table. (Huffman table size is 160 bytes)

Original size	64X64X64 (32 K Bytes)	32X32X32 (4 K Bytes)
Simple coding	1093 bytes	412 bytes
Runlength coding	1250 bytes	481 bytes
Runlength with Huffman	1136 bytes + 160 bytes	392 bytes + 160 bytes
Difference coding	600 bytes	120 bytes

3D Array Compression

We represent voxels as a simple 3D array structure and compresses.

Algorithm 1: Runlength coding

We perform runlength coding directly on 3D representation of voxel data structures.

Algorithm 2: Runlength and Huffman coding

We perform Huffman coding to the output of Algorithm 1.

Measurement

Algorithm	64X64X64	32X32X32
Runlength coding	2320 bytes	627 bytes
Runlength with Huffman	1840 bytes + 160 bytes	440 bytes + 160 bytes

4.2.2. Silhouette Compression

In the final system we directly generate rendered images from silhouettes. Therefore, we may also perform compression of silhouettes in the Phase 2 system.

Algorithm 1: Quadtree

We represent silhouettes as quadtrees.

Algorithm 2: Runlength and Huffman coding

We apply runlength coding to silhouettes and then apply Huffman coding.

Algorithm 3: Huffman coding over differences between runlengths in adjacent scan lines

We convert each scan line to runlengths of 0's and 1's. Then the differences of the runlengths of the adjacent scan lines are taken and then Huffman coding is applied.

Algorithm 4: Runlength and Huffman coding over frame differences

Differences of adjacent frames are taken, then runlength coding is applied, and then Huffman coding is applied.

Algorithm 5: Huffman coding over differences between runlengths in adjacent frames

We convert each scan line of each frame to runlengths of 0s and 1s. Then the differences of runlengths of the corresponding scan lines of adjacent frames are taken and then Huffman coding is applied.

Measurement

Original size	38.4 K Bytes
Quadtree	1000 bytes
Runlength with Huffman coding	1151 bytes + 160 bytes
Huffman over differences of runs of scan lines	525 bytes + 160 bytes
Runlength then Huffman over differences of frames	1840 bytes + 160 bytes
Huffman over differences of runs in between frames	835 bytes + 160 bytes

4.3. Integrated System

The integrated system consists of 8 dual-processor PCs with 500 MHz Pentium III Xeon processors, 16 cameras, 1 dual processor PC with 500 MHz Pentium III processors, and 1 gigabit Ethernet switch which connected 9 PCs together.

Each dual-processor PC captures 2 video streams from 2 cameras, corrects for lens distortion, extracts silhouettes, compresses by bounding box clipping, and then sends the 2 lens distortion corrected and compressed videos and 2 silhouettes to a rendering PC through gigabit Ethernet.

Two video streams and two silhouettes are sent from 8 PCs to the rendering PC. That is a total of 16 video streams and 16 silhouettes per frame. The renderer first waits for all 32 streams to arrive. Then it computes 3 depth images for spanning cameras around the virtual viewpoint, computes the depth image for the virtual viewpoint, and then performs rendering to obtain the image.

We have achieved real time capture, transmission, and render with arbitrary user selected viewpoint at 3 frames per second.

5. Audio

The Phase I effort targeted visual imagery as the primary research and development topic, but also included time for understanding the complexity of adding audio to the visual stimuli. We present our analysis in three parts: general observations, approaches to audio immersion, and off-the-shelf audio support.

5.1. General Observations

Not surprisingly, audio is critical for creating highly immersive experiences. Audio greatly enhances the power of visual stimuli, and person-to-person communication is nearly impossible with audio. To create an immersive audio/video experience, the audio and video must be sufficiently synchronized to avoid confusing human participants in an immersive environment. Through discussions with researchers in the field, we found that the temporal phases of the audio and video signals must differ by no more than 100 milliseconds in order to maintain perceptible synchronization of lips with voice. Practically, this translates into approximately 3 video frames (at the standard video rate of 30 frames/second), so synchronization is achievable. We also found that, once the video frame rate drops significantly below 15 frames per second, the video becomes too choppy to perceive errors in lip synchronization.

5.2. Approaches to Audio Immersion

We define three approaches to add audio to an immersive experience: 2D, conventional 3D, and full 3D. 2D audio involves standard audio capture from one or more microphones and then selection or blending of these signals. Conventional 3D audio involves recording sound sources with one microphone per source, tracking 3D positions of these sources, and then "rendering" the sound field on playback. Full 3D audio recovers all audio sources from an array of microphones and then, like conventional 3D audio, renders the sound field on playback. 2D audio has the lowest computational load, conventional 3D audio has a moderate load, and full 3D has the highest load.

5.2.1. 2D Audio

2D audio involves capture of audio from one or more microphones, transmission of this multi-channel audio stream, and playback of audio through one or more speakers. For capture, microphones can be statically mounted, or can be moving – even attached to people and other objects in the environment. The playback software can switch or mix audio signals from the different microphones to create the signal for each output speaker. In this context, mixing is the traditional operation of adding audio samples together, so computational load is very low. The user can be given a set of switches to control which sound sources are being played back, as well as to control the relative intensities of multiple sources. Depending on the application, simple audio may be sufficient for the types of immersion desired.

5.2.2. Conventional 3D Audio

Like simple audio, conventional 3D audio involves direct capture and manipulation of multi-channel audio. Unlike simple audio, though, audio channels have an associated 3D position. As a result, on playback, the different audio channels can be rendered in 3D. Rendering is usually accomplished by applying a head-related transfer function (HRTF) to the audio signals and spatial locations of the audio sources. The HRTF models many variations in the audio signal received at each ear, including phase and intensity variations.

This type of 3D audio capability works best when the microphones capture sound sources, with no echoes or reverberations. To directly capture sound sources, microphones must be placed closely to those sources. For applications in which human-human communication is the ultimate goal, this would be most easily accomplished by placing wireless microphones on each person. Note that the number of sound sources must be known before capture.

The one missing component in this approach is tracking the 3D location of the microphones. If the immersive environment contains only one person at each site, an approximate 3D location can be derived by computing the center of mass of the voxel model. With more than one person at each location, explicit tracking of each person would be necessary.

5.2.3. Full 3D Audio

The full 3D approach to immersive audio is to compute the sound field in the real environment, then render that field within the viewer's position in the immersive reconstruction. This approach provides the greatest immersive capability, since it can potentially capture all sounds in the environment, not just the sounds instrumented with microphones. To capture sounds, a microphone array would be used to detect sound sources and the positions of those sources. See, for example, the work on the Huge Microphone Array [8], which uses 512 microphones and 128 floating-point signal processors to achieve real-time performance for moderately complex algorithms.

With little standard software support, the full 3D approach is more experimental than either the 2D or the conventional 3D approaches. In addition, full 3D requires many microphones and larger computational power than the other approaches. Finally, the bandwidth requirements to transmit the sound field could be substantially larger than with the previous two approaches.

5.3. Off-the-shelf Audio Support

5.3.1. Hardware and Software Support

Audio is widely supported both by common software programming interfaces as well as by numerous hardware capture and playback option. On the software side, Microsoft provides 3 APIs for audio capture and playback: Wave, MCI, and DirectX. DirectX also supports 3D audio with the DirectSound3D interface, with a built-in model of human audio perception that accounts for phase and attenuation differences between left and right ear as well as sound attenuation as a function of distance between source and listener. Each of these methods also provides mechanisms to synchronize audio to video, which as already noted is important for immersion.

In order to capture audio along with video, we require an interface that maps to our current approach to video capture. In this case, the low-level "wave" interface is the most appropriate, allowing us to use the "waveIn()" function to get fixed-size buffers of fixed-duration audio sequences. The wave interface calls a user-definable function when a buffer fills, allowing continuous capture with little processor utilization. This maps well to our current capture process, which issues video capture instructions once for each video image, then waits for completion with little processor utilization.

5.3.2. Data Rates and Compression

The data rates involved in audio capture are relatively low in comparison to video. CD-quality audio contains 16 bits per sample at a 44.1 kHz sampling rate, for each of two channels. This translates into a data rate of 176.4 KBytes/sec, or about 1.4 Mbits/sec. By comparison, a single video source with YUV 4:2:2 8-bit sampling at 640x480 Y resolution and 30 frames/sec amounts to 18.4 MBytes/sec, or about 147 Mbits/sec.

Even with the relatively low data rate, though, audio compression is an important component in creating widely usable teleportation systems. Several industry standards are available to address audio compression needs. The two most common of these audio coding standards are MPEG/audio and Dolby AC-3.

MPEG/audio defines three operational modes of coding, referred to as Layers I, II, and III. Each layer is increasingly more complex than the previous one, typically providing greater compression at the expense of more computation. Recently, the terminology "MP3" has become a shorthand to refer to MPEG audio, layer III. The target bit rates for each layer are

Layer I	192 Kbits/sec per channel
Layer II	128 Kbits/sec per channel
Layer III	64 Kbits/sec per channel

Dolby AC-3 is another audio coding standard, which has been selected to provide digital surround sound with HDTV broadcasts in the US. This standard explicitly targets multi-channel audio, driven originally by the motion picture industry. Typical applications include 384 Kbits/sec data rates for 5.1-channel audio, and 192 Kbits/sec for two-channel audio. ("5.1-channel audio" denotes 5 full-dynamic-range audio channels along with 1 low-frequency channel, usually to drive a subwoofer in consumer environments.)

Note that these bit rates are for CD-quality audio content, which spans the range of human perception. Much lower data rates are achievable for voice-only audio or for lower-quality sound reproduction.

6. References

[1] S. Laveau and O. D. Faugeras. "3-D Scene Representation as a Collection of Images," In *Proc. of 12th IAPR Intl. Conf. on Pattern Recognitio*n, volume 1, pages 689–691, Jerusalem, Israel, October 1994.

[2] W. Matusik, C. Buehler, R. Raskar, L. McMillan, and S. J. Gortler. " Image-Based Visual Hulls", SIGGRAPH 2000.

[3] L. McMillan. An Image-Based Approach to Three-Dimensional Computer Graphics. Ph.D. Dissertation, University of North Carolina, April 1997. (Also UNC Computer Science Technical Report TR97-013).

[4] P.W. Rander. A Multi-Camera Method for 3D Digitization of Dynamic, Real-World Events. Ph.D. thesis, Carnegie Mellon University, 1998. (Also appears as CMU technical report CMU-RI-TR-98-12.)

[5] P.W. Rander, P.J. Narayanan, and T. Kanade. Recovery of Dynamic Scene Structure from Multiple Image Sequences. Int'l. Conf. On Multisensor Fusion and Integration for Intelligent Systems, 1996.

[6] P.W. Rander, P.J. Narayanan, and T. Kanade. Virtualized Reality: Constructing Time-Varying Virtual Worlds from Real World Events. IEEE Visualization '97, 1997.

[7] H. Saito, S. Baba, M. Kimura, S. Vedula, and T. Kanade. Appearance-Based Virtual View Generation of Temporally-Varying Events from Multi-Camera Images in the 3D Room. 3D Digital Imaging and Modeling (3DIM'99), Oct. 1999. (Also CMU-CS-99-127.)

[8] H.F. Silverman, W.R. Patterson III, and J.L. Flanagan. The Huge Microphone Array (HMA). LEMS Technical Report, Brown University, May 1996.

7. Report Preparers

Peter Rander (412) 682-2862

Norihisa Suzuki (412) 682-2862

Todd Williamson (412) 682-2862

8. Summary

This report summarizes the Phase I developments of Zaxel Systems, Inc on award DAAH01-00-C-R058, Voxel-based Immersive Environments. The overall objective of Phase I was to demonstrate the feasibility of real-time voxel-based immersive visualization by achieving four objectives:

- Demonstrate voxel-based 3D reconstruction at 3 frames per second
- Demonstrate interactive view generation at 3 frames per second
- Determine communication bandwidth requirements and potential compression strategies
- Understand complexity of adding audio (e.g. lip sync)

Zaxel successfully achieved each of these objectives. As part of the resulting system, we developed a collection of additional hardware and software to address technical requirements that must be met for the algorithms to work successfully. These include

- Multi-camera, multi-PC video capture with recording capability: Required synchronization of video across multiple PCs, as well as extensive efforts to support streaming of video to hard disk. Recording video is important for accurately comparing different algorithms and implementations to one another.
- Camera calibration: Developed calibration object, procedure, and software to determine 3D relationships among cameras. This capability is necessary to fuse information from multiple viewpoints.
- Dedicated Studio: Designed a test-bed studio for immersive communication. This test-bed provides the flexibility to test multiple ideas regarding camera placement and background appearance. It also provides valuable experience in designing and constructing more advanced studios.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	5JUL00	FINAL, 9FEB00-7MAY00

4. TITLE AND SUBTITLE
Voxel-based Immersive Environments
Immersive Environments

5. FUNDING NUMBERS
C DAAH01-00-C-R058

6. AUTHOR(S)
Peter Rander
Norihisa Suzuki
Todd Williamson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Zaxel Systems, Inc.
11666 Dawson Dr.
Los Altos Hills, CA 94024

8. PERFORMING ORGANIZATION REPORT NUMBER
ZAX00027

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
Attn: ISO (Mr. Ward C. Page)
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
---

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT (see Section 5.3b of this solicitation)
Distribution authorized to U.S. Government Agencies only; contains proprietary information.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report summarizes the Phase 1 developments of Zaxel Systems, Inc. on award DAAH01-00-C-R058, Voxel-based Immersive Environments. The overall objective of Phase 1 was to demonstrate the feasibility of real-time voxel-besed immersive visualization by achieving four objectives: Demonstrate voxel-based 3D reconstruction at 3 frames per second; Demonstrate interactive view generation at 3 frames per second; Determine communication bandwidth requirements and potential compression strategies; Understand complexity of adding audio. Zaxel successfully achieved each of these objectives. As part of the resulting system, we developed a collection of additional hardware and software to address technical requirements that must be met for the algorithms to work successfully. These include: Multi-camera, multi-PC video capture with recording capability: Required synchronization of video across multiple PCs, as well as extensive efforts to support streaming of video to hard disk. Recording video is important for accurately comparing different algorithms and implementations to one another; Camera calibration: Developed calibration object, procedure, and software to determine 3D relationships among cameras. This capability is necessary to fuse information from multiple viewpoints; Dedicated Studio: Designed a test-bed studio for immersive communication.

14. SUBJECT TERMS	15. NUMBER OF PAGES
SBIR Report, Virtual Immersion, Teleportation, Remote Collaboration, 3D Pixelization, Real-Time Systems, Data Compression, Video Interpolation, 3D Modeling	33
	16. PRICE CODE: ---

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

Standard form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Zaxel Systems, Inc.

11666 Dawson Drive
Los Altos Hills, California 94024

Ten 40th Street
Pittsburgh, Pennsylvania 15201
Tel: 412-682-2862
Fax: 412-682-2856

www.zaxel.com

July 7, 2000

Defense Technical Information Center
ATTN: Acquisitions/DTIC-OCP, Rm-815
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

To Whom It May Concern:,

On July 5, we submitted our Final Report for DAAH01-00-C-R058, Voxel-based Immersive Environments. I would like to replace our Report Documentation Page with the one I am submitting in this mailing. I have changed block 12a. I greatly appreciate your effort in switching pages.

Please contact me at (412) 682-2862 if you have any questions in this matter.

Sincerely,

Carolyn S. Ludwig
Business Manager
Zaxel Systems, Inc.